CSC406Assignment#2 Due on or before March 4, 2015 raj

Implement the following algorithms:

1. Topological sort.
2. Kruskal’s algorithm for finding a MST.

This assignment needs the implementation of undirected and weighted ( > 0 ) graphs. Such a graph must be implemented in both matrix and list representations. Must use Java’s PriorityQueue to implement a heap. This project needs to be built on top of the first assignment.

As usual pay attention to the following additional requirements. More may be added to this list after I look through your first assignment.

1. Clearly highlight the Assignment#2 code.
2. Don’t maintain a collection of edges. Maintaining such a collection is yet another way of representing a graph. We want to use only two rerpesentations of a graph, namely, adjacency matrix and adjacency list representations. Storing the edges within each of these representations is redundant, and redundancy can lead to data integrity problems in addition to increasing the memory requirement by one hundred percent.
3. Correctness of the code alone is not sufficient for full credit. Be aware of complexity of your code.
4. Using an arrayList is generally costlier than using an array. So, if you know the size of the collection then use an array for the collection instead of an arrayList.
5. Use Iterators wherever you have to step through each element of a collection from beginning to end. Using an iterator is very simple. Here is a skeleton code.

ArrayList<Node> aList;

Iterator<Node> ite = aList.iterator( ); //note the case for i in iterator

// repeat if there are more elements in the collection

while (ite.hasNext( ) ) {

Node node = ite.next( ); //get the next element from the collection

… //process node.

}

1. Provide appropriate documentation to explain complex code.
2. When operating on a collection (such as an ArrayList ) write your own method only if an equivalent method is not available in the API listing for that collection.
3. While implementing Topological Sort don’t remove the edge from the graph every time through the loop. It is generally desirable to preserve the input as much as possible. So, don’t remove the edge from the graph; instead, simply adjust the inDegree.
4. Operations such as search or delete elements from a collection involves comparison for equality between objects. So, don’t forget to add an equals method to the class whose instances may have to be compared.

For example, given the collection ArrrayList<Node> aList;

To perfrom a search on aList using contains method, aList.contains(n), you must have an equals method in the Node class with the following signature.

public boolean equal(Object node) { //note the type of the parameter

Node n1 = (Node) node; // cast the parameter before use

….

}

1. In all versions of your program include your name, course#, semester, and assignment# along with date assigned and date submitted.
2. Seek immediate help if you need help in understanding/implementing your project.